

Лабораторная работа 6 «Функции. Рекурсивные алгоритмы»

Цель работы: познакомиться с наиболее востребованными стандартными функциями Python, изучить синтаксис объявления и использования пользовательских функций; познакомиться с рекурсивными алгоритмами; научиться проектировать и разбивать большую программу на мелкие фрагменты (функции); закрепить знания на примере разработки интерактивных приложений.

Основное задание

1. Рекурсивно описать функцию $f(x, n)$, вычисляющую $x^n/n!$ при любом действительном x и любом неотрицательном целом n .
2. Рекурсивно описать функцию $row(x, n)$, вычисляющую x^n для любого действительного $x \neq 0$ и любого целого n .
3. Реализовать функцию, которая вычисляет N -й элемент ряда Фибоначчи с использованием рекурсивного алгоритма. На базе данной функции разработать программу, которая должна предлагать пользователю следующие возможности:
 - 1) вывод конкретного элемента последовательности;
 - 2) вывод всех элементов до указанного пользователем элемента;
 - 3) вывод части последовательности, значение последнего элемента которой не превосходит введённого пользователем значения.

Индивидуальное задание

Модернизировать программы и оптимизировать алгоритмы решения основных и дополнительных заданий лабораторной работы 5 (часть 2), используя функции (возможно рекурсию).

Дополнительное задание*

1. Найти все натуральные числа, не превосходящие N , сумма цифр каждого из которых в некоторой степени дает это число (например: $7^4 = 2401$, $8^3 = 512$, $9^2 = 81$ и т.д.).
2. Напечатать N автоморфных чисел (числа, совпадающие с младшими цифрами своего квадрата, например: 25 и 625).
3. Найти N троек чисел Пифагора (натуральные числа a , b и c называются числами Пифагора, если выполняется условие $a^2 + b^2 = c^2$).
4. Описать рекурсивную функцию $equals(N, S)$ (где N и S – неотрицательные целые числа), которая проверяет, совпадает ли сумма цифр в десятичной записи числа N со значением S (например: $equals(1234567, 28) = True$, $equals(10, 7) = False$).
5. Рекурсивно описать функцию $divs(N)$ для подсчета количества всех делителей заданного целого числа N ($N > 1$), без учета делителей 1 и N (например: $divs(7) = 0$, $divs(10) = 2$, $divs(16) = 3$ и так далее).
6. Задача о Ханойской башне (*Tower of Hanoi*). Имеется три стержня А, В, С, на один из которых нанизаны N колец, причем кольца отличаются размером и лежат меньшее на большем. Требуется перенести пирамиду из N колец с одного стержня на другой за наименьшее число ходов. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее.

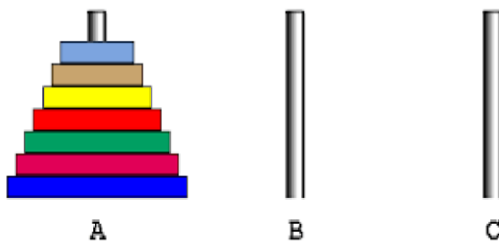


Рис. 6.7. Задача о Ханойской башне

Например, для функции `hanoi(3, 'A', 'B', 'C')`, где 3 – число колец, А – стержень-источник, В – стержень-приемник, С – имя временного стержня, должен получиться ответ, как на рисунке 6.8:

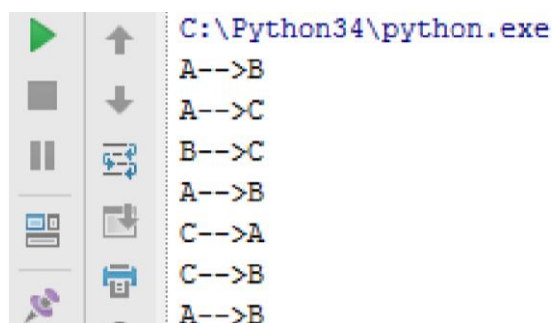


Рис. 6.8. Решение задачи о Ханойской башне



Для тех, кто хочет закрепить свои знания и навыки для решения задач с помощью рекурсий, могут быть полезны ресурсы:

<http://server.179.ru/tasks/training/recursion.html>,

<https://ru.wikibooks.org/wiki/>

Требования к выполнению

1. Программа должна обязательно быть снабжена комментариями, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчика и дату разработки.

2. В программах по необходимости предусмотреть возможность её повторного выполнения, а также защиты от ввода некорректных пользовательских данных (предусмотреть «защиту от дурака»). Для этого рекомендуется разработать отдельные функции;

3. При разработке программ рекомендуется придерживаться принципа единственной ответственности, то есть функции должны проектироваться и реализовываться таким образом, чтобы они были менее завязаны с другими функциями при своей работе.

4. Каждая программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом.

Контрольные вопросы

1. Что такое структурное программирование?
2. Что такое функция? Как описывается функция в *Python*?
3. Зачем нужны функции?
4. Для чего используется оператор *return* в функциях? Как вернуть из функции несколько значений?
5. Чем формальные параметры отличаются от фактических?
6. Что такое позиционные параметры?
7. Что такое параметры по умолчанию?
8. Чем отличается глобальная переменная от локальной?
9. Зачем применяются рекурсивные алгоритмы?
10. В чём заключается смысл принципа единственной ответственности (*Single Responsibility Principle*) при грамотной разработке единиц программного кода (в частности, функций)?

Функции в языке Python. Рекурсивные алгоритмы. Модули

ФУНКЦИИ. ВЫЗОВ ФУНКЦИИ

В контексте программирования *функцией* (*function*) называется хранимая последовательность инструкций, предназначенная для решения определенной задачи. Основными параметрами функции являются:

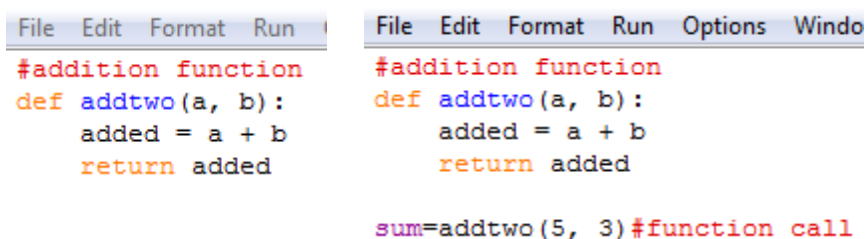
- имя функции;
- тело функции;
- передаваемые параметры (аргументы);
- возвращаемые параметры (результат).

В *Python* предоставляется возможность создавать свои собственные (пользовательские) функции, а также работать со встроенными стандартными функциями (функции ввода/вывода данных, функции преобразования типов, математические функции модуля *math*, функции генерации псевдослучайных функций модуля *random*).

Правила наименования функций такие же, как для переменных, например, нельзя использовать зарезервированные слова в качестве имен функций. Имена функций и переменных не должны совпадать.

Первая строка определения функции называется *заголовком* (*header*), оставшаяся часть – *телом* (*body*) функции. Заголовок заканчивается двоеточием, тело функции имеет отступ. Тело функции может содержать любое количество инструкций. Инструкции внутри функции не получают управления, пока функция не будет вызвана. Для возврата результата функции, используется инструкция *return*. Вызвать функцию (*function call*) можно, обратившись к ней по имени.

Например, на рисунке 6.1 представлена функция *addtwo()*, имеющая два аргумента, и обращение к ней. Указанная функция складывает два числа, и возвращает результат. Для вызова этой функции ей нужно передать два параметра. Возвращаемый параметр функции – переменная *added*, в которую записывается сумма аргументов.



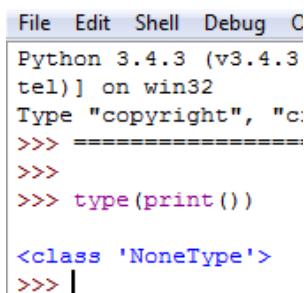
```
File Edit Format Run
#addition function
def addtwo(a, b):
    added = a + b
    return added

sum=addtwo(5, 3)#function call
```

Рис. 6.1. Функция *addtwo* и ее вызов

Пустые скобки после имени функции указывают на то, что функция не требует аргументов, например, функция *random()*. Если попытаться присвоить результат выполнения такой функции переменной, вернется специальное значение, называемое *None*.

Существуют функции, не возвращающие результат, например, функция *print()* (рисунок 6.2).

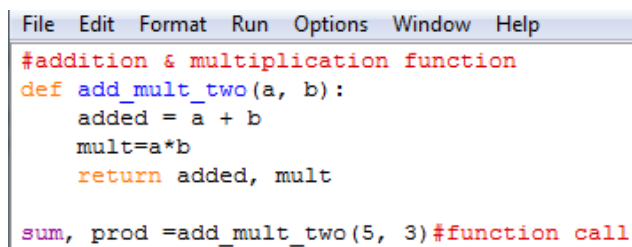


```
File Edit Shell Debug C
Python 3.4.3 (v3.4.3
tel)] on win32
Type "copyright", "c:
>>> =====
>>>
>>> type(print())

<class 'NoneType'>
>>> |
```

Рис. 6.2. Функция *print()*

На рисунке 6.3 приведен пример функции *add_mult_two(a, b)*, возвращающей несколько значений.



```
File Edit Format Run Options Window Help
#addition & multiplication function
def add_mult_two(a, b):
    added = a + b
    mult=a*b
    return added, mult

sum, prod =add_mult_two(5, 3)#function call
```

Рисунок 6.3 – Функция `add_mult_two(a, b)`

ПОТОК ИСПОЛНЕНИЯ

Для того чтобы убедиться, что функция определяется до ее первого использования, нужно знать порядок, в котором выполняются инструкции, он называется *поток исполнения (flow of execution)*. Исполнение обычно начинается с первой инструкции программы.

Вызов функции подобен обходному пути в потоке исполнения. Вместо того чтобы перейти к следующей инструкции, поток переходит в тело функции, выполняет все инструкции внутри тела, и затем возвращается обратно в то место, которое он покинул в момент вызова функции. При этом в процессе вызова одной функции, программа может выполнять инструкции из другой.

ПРИНЦИП ЕДИНСТВЕННОЙ ОТВЕТСТВЕННОСТИ

Программу рекомендуется разбивать на функции, поскольку:

- создание новой функции предоставляет возможность присвоить имя группе инструкций, что позволит упростить чтение, понимание и отладку программы;
- функции позволяют сократить код программы, благодаря ликвидации повторяющихся участков кода;
- разбиение длинной программы на функции позволяет одновременно отлаживать отдельные части, а затем собрать их в единое целое;
- хорошо спроектированная функция может использоваться в других программах.

Принцип единственной ответственности (The Single Responsibility Principle) декларирует разграничение ответственности между функциями: за решение одной конкретной задачи должна отвечать одна функция.

Использование принципа единственной ответственности при работе с функциями является хорошим стилем программирования.

Более детально принцип рассматривается при изучении основ объектно-ориентированного программирования.

Пример выполнения задания для вычисления элементов ряда Фибоначчи

Пусть необходимо реализовать функцию, которая вычисляет первые N -элементов ряда Фибоначчи, а также написать программу для её тестирования. Реализация представлена на рисунке 6.4.

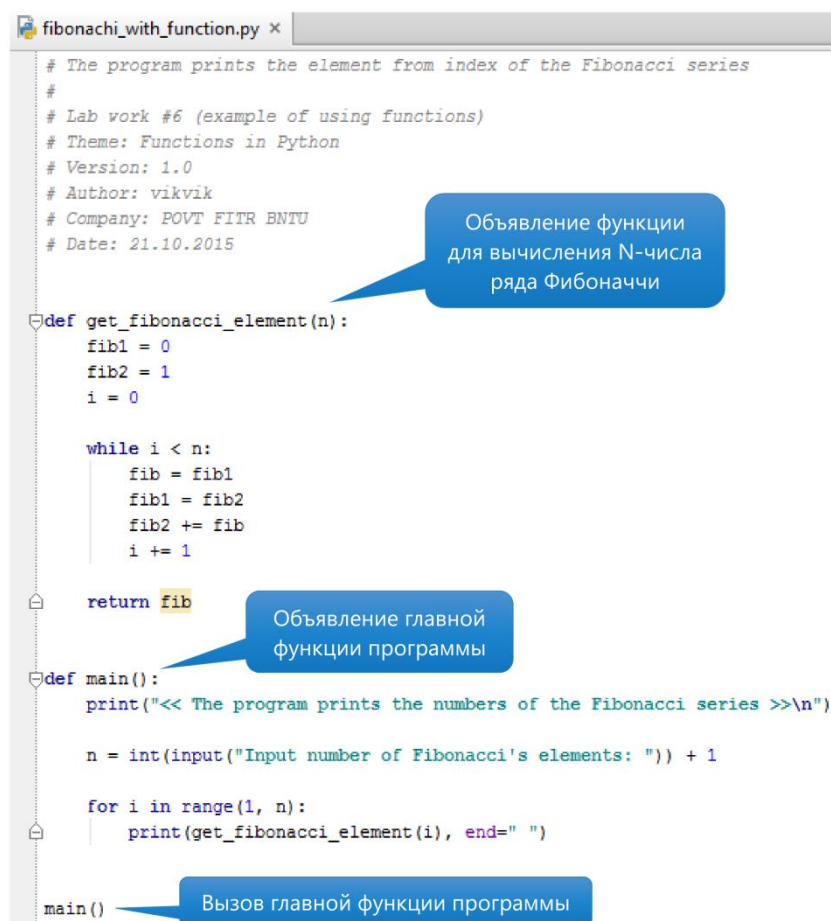
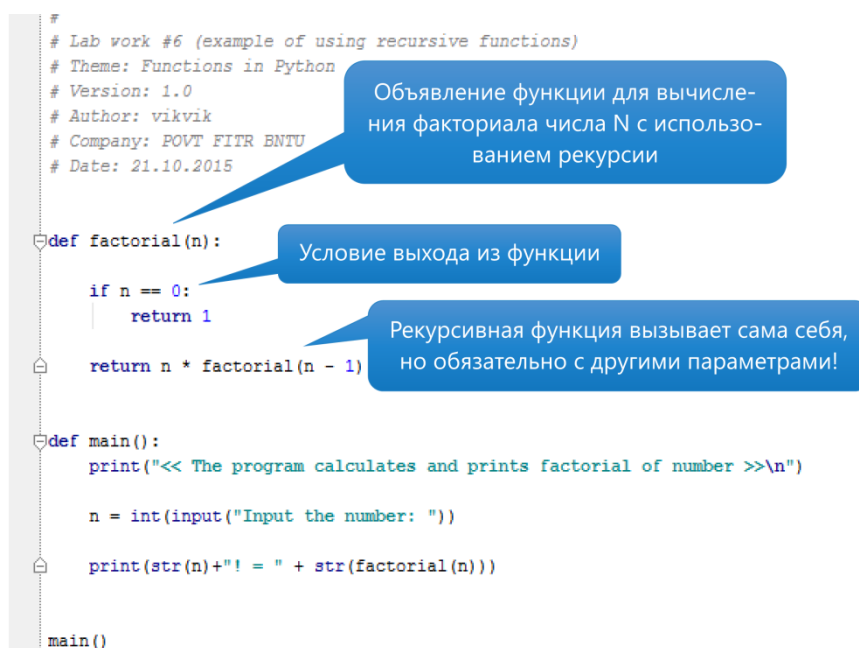


Рис. 6.4. Код программы решения задания

Пример выполнения задания с использованием рекурсивного подхода

Необходимо разработать программу, которая вычисляет факториал числа N с использованием двух способов: итеративного и рекурсивного. Этапы реализации представлены на рисунках 6.9 – 6.15.

Рис. 6.5. Вычисление факториала числа N рекурсивным методом

```
def factorial(n):  
    p = 1  
    for i in range(1, n + 1):  
        p *= i  
    return p  
  
def main():  
    print("<< The program calculates and prints factorial of number >>\n")  
    n = int(input("Input the number: "))  
    print(str(n)+"! = " + str(factorial(n)))  
  
main()
```

Рис. 6.6. Вычисление факториала итерационным методом